Page Faults

Page Faults — handling

Cause: access to page currently not present in main memory
→ exception, invoking OS

Process:
— OS checks access validity (requiring additional info)
— get empty frame
— load contents of requested page from disk into frame
— adapt page table
— set present bit of respective entry
— restart instruction causing page fault

Page Faults — latency

fault rate $0 \leq p \leq 1$
— $p = 0$: no page faults
— $p = 1$: every reference leads to page fault

effective access time (EAT):
EAT $= (1 - p) *$ memory access $+ p *$ (PF overhead $+$ PF service time $+$ restart overhead)

Page Faults — performance impact

memory access time: 200ns

average page fault service time: 8ms

$\rightsquigarrow$ 1:1000 access-page-fault-rate $\rightarrow$ EAT $= 8.2\mu s \Rightarrow$ *slowdown by factor 40*!

Page Faults — challenges

what to eject?
— how to allocate frames among processes?
— which particular process's pages to keep in memory?
— see *page frame allocation*

what to fetch?
— what if block size $\neq$ page size?
— just one page needed? prefetch more?

process resumption?
— need to save state + resume
— process might have been in middle of instruction

Page Faults — what to fetch?

bring in page causing fault

pre-fetch sourrounding pages?
— reading two disk blocks is approximately as fast as reading one
— as long as there is no track/head switch, seek (disk) time dominates
— application exhibits spatial locality = big win

pre-zero pages?
— don't want to leak information between processes
— need 0-filled pages for stack, heap, .bss, …
— *zero on demand*?
— keep pool of 0-pages filled in background when CPU is idle?

Page Faults — process resumption?

hardware provides info about page fault
(intel: `%cr2` contains faulting virtual address)

context: OS needs to figure out fault context:
— read or write?
— instruction fetch?
— user access to kernel memory?

idempotent instructions: easy:
— re-do load/store instructions
— re-execute instructions accessing only one address

complex instructions: must be re-started
— some CISC instructions are hard to restart (e.g., block move of overlapping areas)
— *solutions*:
· touch relevant pages before operation starts
· keep modified data in registers $\rightarrow$ page faults can't take place
· design ISA such that complex operations can execute partially $\rightarrow$ consistent page fault state